Python Session 2

This session:

1. User Input
2. Importing modules
3. For Loops
4. While Loops
5. Functions

User Input

The `input()` function allows you to input data after the program has started running

This program uses input to ask what your name is

In [ ]:
```
age = input('What is your age? ')
print('Hello, I am {} yo'.format(age))
```

Exercise 2.1: Write a program that asks two questions using `input()` then prints the values that were entered. You can choose any questions that you want.

Example:

In [ ]:
```
fruit = input('What fruit do you like? ')
veg = input('What veg do you like? ')

print('You like {} and you like {}'.format(fruit, veg))
```

The `int()` function converts string value into integer values:

In [ ]:
```
apples_string = '12'
total_apples = int(apples_string) + 5

print(total_apples)
```

The `input()` always returns a string value. You can convert this string value to an integer with `int()` :

In [ ]:
```
purchased_apples = input('How many apples did you buy? ')
print(type(purchased_apples))
total_apples = int(purchased_apples) + 5

print(total_apples)

type()
```

Exercise 2.2: You have friends at your house for dinner and you've accidentally burnt the lasagne. Time to order pizza.

Write a program calculate how many pizzas you need to feed you and your friends

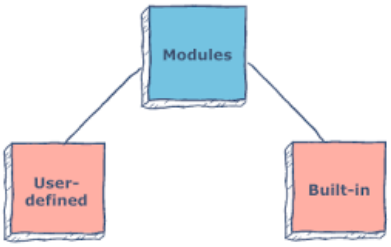
In [ ]:
```
friends = input('How many friends? ')
pizzas = int(friends) * 0.5

print('You need {} pizzas for {} friends'.format(pizzas, friends))
```

Solution

In [ ]:
```
friends = int(input('How many friends are at your house? '))
pizzas = friends * 0.5

print('You need {} pizzas for {} friends'.format(pizzas, friends))
```

Python Modules

Module: Code that someone else has written that you can reuse in your programs



Modules are imported into your Python programs:

`import `

also

`from import `

Examples of python modules:

math --> mathematical functions

datetime --> date and time value manipulation

timeit --> time the execution of small blocks of Python code

re --> regular expressions (pattern search)

copy --> duplicating objects

- we would learn more about various modules throghout the course
- let's review some examples based on **datetime** (we only want to review some examples)

Problem Solving (with datetime)

Python datetime package documentation: https://docs.python.org/3/library/datetime.html (https://docs.python.org/3/library/datetime.html)

After importing a module you can use the module's functions. For example, we can use the function that can tell us exactly what day and time it is now.

In [ ]:
```
import datetime

x = datetime.datetime.now()
print(x)
```

Can you think of any examples where this function would be useful?

Have you ever heard about a **timestamp** ?

Datetime package has many useful functions

- We can find out date, time, timedate and even timedelta with this module
- We can create dates by converting text into date objects
- We can perform calculations to compute dates in the past and in the future

Creating Date Objects

In [ ]:
```
Can you think of any examples where this function would be useful?

Have you ever heard about a timestamp ?
```

Formatting date objects

In [ ]:
```
import datetime

my_date = datetime.date(2020, 12, 31)

print(my_date.strftime("%d-%b-%Y"))
```

Character codes examples:

- %a: Returns the first three characters of the weekday, e.g. Wed.
- %A: Returns the full name of the weekday, e.g. Wednesday.
- %B: Returns the full name of the month, e.g. September.
- %w: Returns the weekday as a number, from 0 to 6, with Sunday being 0.
- %m: Returns the month as a number, from 01 to 12.
- %p: Returns AM/PM for time.
- %f: Returns microsecond from 000000 to 999999.
- %Z: Returns the timezone.
- %Y: Returns the year in four digit format
- %b: Returns the first three characters of the month name.
- %d: Returns day of the month, from 1 to 31.
- %Y: Returns the year in four-digit format.
- %H: Returns the hour.
- %M: Returns the minute, from 00 to 59.
- %S: Returns the second, from 00 to 59.

For Loops

`for` loop: allows you to repeat a block of code multiple times

In [ ]:
```
# think of range(5) as a collection of five digits in a box: 0,1,2,3,4 starting from a 0
# the FOR loop would iterate through every digit

for number in range(5):
    print(number)
```

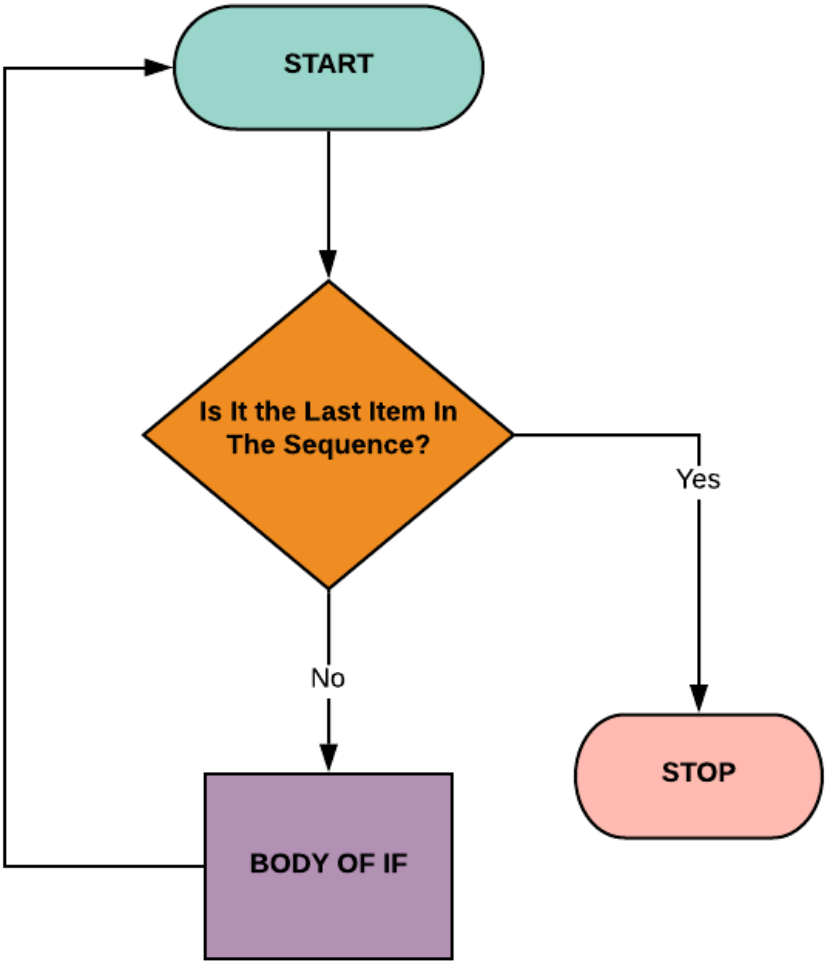
In [ ]:
```
# think of any word as a collection of characters that we can iterte over

for character in 'Banana':
    print(character)
```

In [ ]:
```
# think of any word as a collection of characters that we can iterte over

for character in 'Banana':
#     print('<' + character + '>') # this action will be repeated for each character in the word!
```

In [ ]:
```
# think of this examples as a box with words, we are doing an action for each word
#NB: this is called a 'list' but we will learn about it later in the course, this is only an example fo
r now.

for name in ['Mary', 'Ranjit', 'Fatima']:
    print(name)
```

FOR LOOP

- A for loop is used to iterate over sequences (a collection of items).
- And not only just the sequences but any iterable object can also be traversed.
- The execution will start and look for the first item in the sequence.
- After executing the statements in the block, it will look for the next item and the process will continue until the the last item is reached.

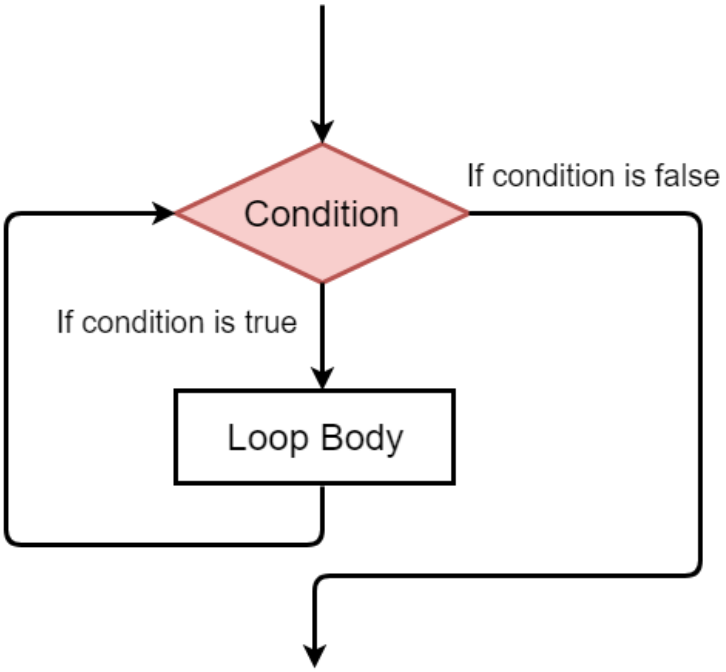


In [ ]:
```
for number in range(5): # 0,1,2,3,4
    print("executing FOR LOOP - run No {}".format(number + 1))
```

In [ ]:
```
total = 0
print("*** This statement is OUTSIDE THE LOOP")
print("Before the loop executes, our TOTAL is equal to  = ", total, '\n')
print("-------------------------------------------------------")

for number in range(3): # remember --> 0, 1, 2

    print("This is loop execution for digit: " + str(number) + " inside the loop \n")
    print("Updating total... (+ 1) \n")

    total  = total + 1 # every time the loop executes, we add 1 to the total


print("-------------------------------------------------------")
print("***This statementWe is OUTSIDE the loop now")
print("The final TOTAL value is: " + str(total))
```

While Loop

A while loop in python is used to iterate over a block of code or statements as long as the test expression is true.



`WHILE loop`

- In case of a while loop a user does not know beforehand how many iterations are going to take place.
- Beware of infinite while loops - they execute infinite times i we don't specify correct condition
- If the loop is running infinitely and never stops, we would 'blow' our memory usage and the program would encounter an error.

Example:

Due to social disctancing, only 10 people are allowed to be inside a shot at the same time. This program invites people in the queue to come in while we have some capacity.

In [ ]:
```
store_capacity = 5  #

while store_capacity > 0:
   print('Please come in. Spaces available: ' + str(store_capacity))
   store_capacity =  store_capacity - 1

print("\nPlease wait for someone to exit the store.")
```

In [ ]:
```
# Example INFINITE 'while loop' that runs forever until the memory is 'blown'

store_capacity = 10

while store_capacity > 0:
   print('Please come in. Spaces available: ' + str(store_capacity))
   # store_capacity =  store_capacity - 1 ---> imagine that we forgot to add this logic!!!


print("\nPlease wait for someone to exit the store.")
```

`While loop` is also very useful for the programming algorithms and program flow control.

We will practice more examples of **while loop** when we learn about IF/ELSE logic (as they go hand in hand)

Functions

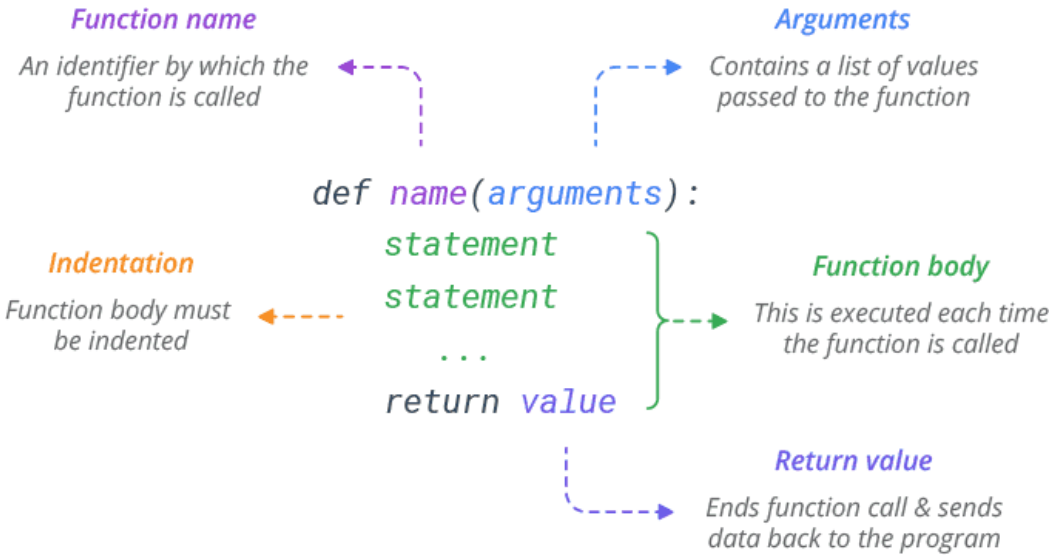
Function: is a reusable block of code that contains one or more Python statements and used for performing a specific task.

Why use function in Python?

1. **Code re-usability**: it is better to wrap 10 lines of code in a function and just call the function wherever needed, as opposed to writing those 10 identical lines every time you perform that task.

1. **Improves Readability**: By using functions for frequent tasks you make your code structured and readable.

1. **Avoid redundancy**: When you no longer repeat the same lines of code throughout the code and use functions in places of those, you actually avoiding the redundancy that you may have created by not using functions.



Create a Function

- To define a Python function, use `def` keyword.
- Here's the simplest possible function that prints a greeting

In [ ]:
```
def hello():
    print('Hello, class!')
```

Call a Function

- The `def` statement only creates a function but does not call it.
- After the `def` has run, you can can call (in other words run) the function by adding parentheses after its name.

In [ ]:
```
def hello():
    print('Hello, class ***!')

hello()
```

Pass Arguments

- You can send information to a function by passing values, known as arguments.
- Arguments are declared after the function name in parentheses.
- When you call a function with arguments, the values of those arguments are copied to their corresponding parameters inside the function.

In [ ]:
```
# Pass single argument to a function
def hello(name):
    print('Hello,', name)

hello('Maria')
hello('Kim')
hello('Olya')
```

In [ ]:
```
# Pass multiple arguments

def some_function(name, job):
    print(name, 'is a', job)

some_function('developer', 'Fiona')
```

Function Arguments

Types of arguments

Python supports multiple types of arguments in the function definition:

- Positional Arguments
- Keyword Arguments
- Default Arguments
- Variable Length Positional Arguments (*args)
- Variable Length Keyword Arguments (**kwargs)

Positional Arguments

- Positional arguments values are copied to their corresponding parameters in order.
- You must to pass arguments in the order in which they are defined.

In [ ]:
```
# Correct arguments order

def some_function(name, job):
    print(name, 'is a', job)

some_function('Fiona', 'developer')
```

In [ ]:
```
# Wrong arguments order

def some_function(name, job):
    print(name, 'is a', job)

some_function('developer', 'Fiona')
```

Keyword Arguments

- you can pass arguments using the names of their corresponding parameters.
- in this case, the order of the arguments no longer matters.
- you can combine positional and keyword arguments in a single call.
- if you do so, specify the positional arguments before keyword arguments.

In [ ]:
```
def some_function(name, job):
    print(name, 'is a', job)

some_function(job='developer', name='Fiona')

some_function(name='Fiona', job='developer')
```

Default Arguments

- You can specify default values for arguments when defining a function.
- The default value is used IF the function is called without a corresponding argument.

In [ ]:
```
def some_function(name, job='developer'):
    print(name, 'is a', job)

some_function('Fiona')

some_function('Fiona', 'manager')
```

Returning Values from Function

Return Value

- To return a value from a function, simply use a return statement.
- Once a return statement is executed, nothing else in the function body is executed.
- Remember! a python function ALWAYS returns a value.
- So, if you do not include any return statement, it automatically returns None.

In [ ]:
```
# Using return statement

def sum(x, y):
    return x + y


result = sum(10, 15)

print("result is: {}".format(result))
```

In [ ]:
```
# Without return statement

def sum(x, y):
    print(x + y)

result = sum(10, 15)
print("result is: {}".format(result))
```

In [ ]:
```
# Without return statement

def sum(x, y):
    x + y

result = sum(10, 15)
print("result is: {}".format(result))
```

Exercise 2.6: Complete the function to return the area of a circle

Use the comments to help you

In [ ]:
```
def circle_area():  # add the radius argument inside the brackets
    area = 3.14 * (radius ** 2)
    # add return statement


circle_1 =  circle_area(10)

print(circle_1)
```

Solution

In [ ]:
```
def circle_area(radius):
    area = 3.14 * (radius ** 2)
    return area

area = circle_area(9)

print(area)
```

What's the outcome of this?

In [ ]:
```
def days_in_hours(days):
    hours = days * 24
    return hours

print(days_in_hours(10))
```

Combine FOR loop and FUNCTION

In [ ]:
```
def times_two(num):
    result = num * 2
    return result


for number in range(3): # 0,1,2
    calc_res = times_two(number)
    print(calc_res)
```

Homework: Session 2 homework questions in your student guide